Gazi Üniv. Müh. Mim. Fak. Der.
Cilt 28, No 3, 545-554, 2013

Journal of the Faculty of Engineering and Architecture of Gazi University
Vol 28, No 3, 545-554, 2013

ÇOK ÇEKİRDEKLİ İŞLEMCİLERDE SET-BAZLI DİNAMİK ÖNBELLEK BÖLÜNMESİ

Gürhan KÜÇÜK*, İsa Ahmet GÜNEY*
* Yeditepe Üniversitesi, Mühendislik ve Mimarlık Fakültesi, Bilgisayar Mühendisliği Bölümü, İstanbul
gkucuk@cse.yeditepe.edu.tr, iguney@cse.yeditepe.edu.tr

(Geliş/Received: 04.03.2013; Kabul/Accepted: 18.07.2013)

ÖZET

Günümüzde çok çekirdekli işlemciler, çekirdek-dışı bellek erişimlerindeki gecikmeleri azaltmak için çekirdekler tarafından paylaşılabilen bir son seviye önbellek içermektedir. Ancak, çekirdekler üzerinde paralel olarak çalışan uygulamalar çok fazla sayıda önbellek çatışmasına yol açarak bu tip önbelleklerden elde edilebilecek faydaları kısıtlayabilmektedir. Literatürde bu önbellek seviyesini bölümleyerek her uygulamaya özel bir alan yaratan ve sonuçta önbellek çatışmalarını azaltmaya çalışan birçok çalışma mevcuttur. Genelde bu çalışmalar her bir çekirdeğe ihtiyacına uygun sayıda önbellek yolu atamaya odaklanmıştır. Bunun yanında son zamanlarda önerilen set bazında önbellek bölümlenmesi öneren çalışmalar da mevcuttur. Set-bazlı bölümlemenin yol-bazlı bölümlemeye göre birtakım avantajları bulunmaktadır. Bu çalışma, son seviye önbellek yapılarını set-bazlı olarak bölerek işlemci başarımının iyileştirilmesini hedeflemektedir. Bölümleme kararları, donanım yardımı ile periyodik olarak toplanan, çalışan uygulamalara ait çalışma-anı istatistikleri yardımıyla verilmektedir.

Anahtar Kelimeler:Eş-zamanlı Çoklu İş Parçacıklı İşlemciler, Dinamik Önbellek Bölümleme

SET-BASED DYNAMIC CACHE PARTITIONING ON CHIP MULTIPROCESSORS

ABSTRACT

Today, most of the chip multiprocessor architectures utilize a shared last level cache to reduce the off-chip memory delay. Benefit from such a cache may be very limited due to cache conflicts caused by applications running in parallel. In the literature, there are numerous studies that try to reduce cache conflicts by partitioning this cache level and allocating dedicated cache areas to each application. These studies generally focus on policies dedicating an appropriate number of ways to each core. There has also been recent studies suggesting set-based cache partitioning. Set-based partitioning has a number of advantages over way-based partitioning. This study aims to improve the processor performance by using a mechanism to dynamically partition the cache based on sets. The resizing decisions for partitions are made according to statistics collected at periodic intervals.

Keywords: Simultaneous Multi Threaded Processors, Dynamic Cache Partitioning

1.GİRİŞ (INTRODUCTION)

Birden fazla sayıda çekirdeği bir işlemci paketi içine sığdırmanın başlıca hedefi birden fazla uygulamayı paralel olarak çalıştırarak tek çekirdekli işlemcilere göre daha yüksek iş çıktısı elde etmektir. Günümüzde tipik bir çok çekirdekli işlemci (ÇÇİ) mimarisi her bir çekirdeğe ait birinci seviye (ve bazen ikinci seviye) önbellek sunmaktadır. Veri akımlarındaki zamansal mevki (temporal locality) kaynaklı erişimlerin çoğu da bu önbellek seviyesinde yakalanmaktadır. Ancak, özellikle birinci seviye önbellekler, hızlı erişim endişesi ile tasarlanmakta ve gerçeklenmekte, sonuç olarak da veri akımlarındaki uzamsal mevki (spatial locality) kaynaklı erişimleri yakalayabilecek önbellek blok boyutları ve kapasitelerine sahip olamamaktadırlar. Genelde, uzamsal mevki kaynaklı erişimleri yakalayabilmek için tüm çekirdekler tarafından paylaşılan, üst seviyelere göre çok daha yüksek kapasiteye sahip bir son (ikinci veya üçüncü) seviye önbellek kullanılmaktadır. Bu tip bir önbellek, özellikle üretici/tüketici tarzı iş parçacıkları (thread) içeren uygulamalar için yüksek hızlı bellek erişimi vaat etmektedir. Ancak, çekirdekler üzerinde çalışan

uygulamaların birbirinden bağımsız olmaları durumunda paylaşılan bir önbellek daha kötü bellek erişim zamanları da sunabilir. Bunun en tipik örneği, son seviye önbelleği paylaşan birden fazla uygulamanın birbirlerinden karşılıklı önbellek satırları çalarak önbelleği kullanılmaz hale getirmesidir. Bu durumda uygulamalar, tek başlarına çalışmalarından daha kötü bir başarım sergileyecektir.

Son seviye önbelleğin paylaşılmaması ve her bir çekirdek için bir parçasının ayrılması da iyi bir çözüm olarak görülmemektedir. Çekirdeklere atanmış tüm uygulamaların aynı boyutta önbellek ihtiyacı göstermeyeceği kesindir. Bu gibi durumlarda önbellek ihtiyacı çok yüksek olan bir uygulama son seviye önbelleğin sadece kendisine ayrılan, kısıtlı boyuttaki kısmına erişebilecek ve bir diğer uygulama önbelleğe ihtiyaç duymasa da önbelleğin bir parçasını elinde tutacaktır. Sonuç olarak, paylaşılan veya paylaşılmayan önbellek organizasyonlarının çok çekirdekli işlemcilerin başarımına katkısı sınırlı kalmaktadır. Çalışan uygulamaların tipleri en iyi önbellek organizasyonu konusunda ipuçları vermektedir, ve çalışma anında çalışan iş parçacıklarına ait toplanan istatistikler ışığında dinamik olarak önbelleği çekirdekler arasında paylaştıracak mekanizmalara ihtiyaç bulunmaktadır.

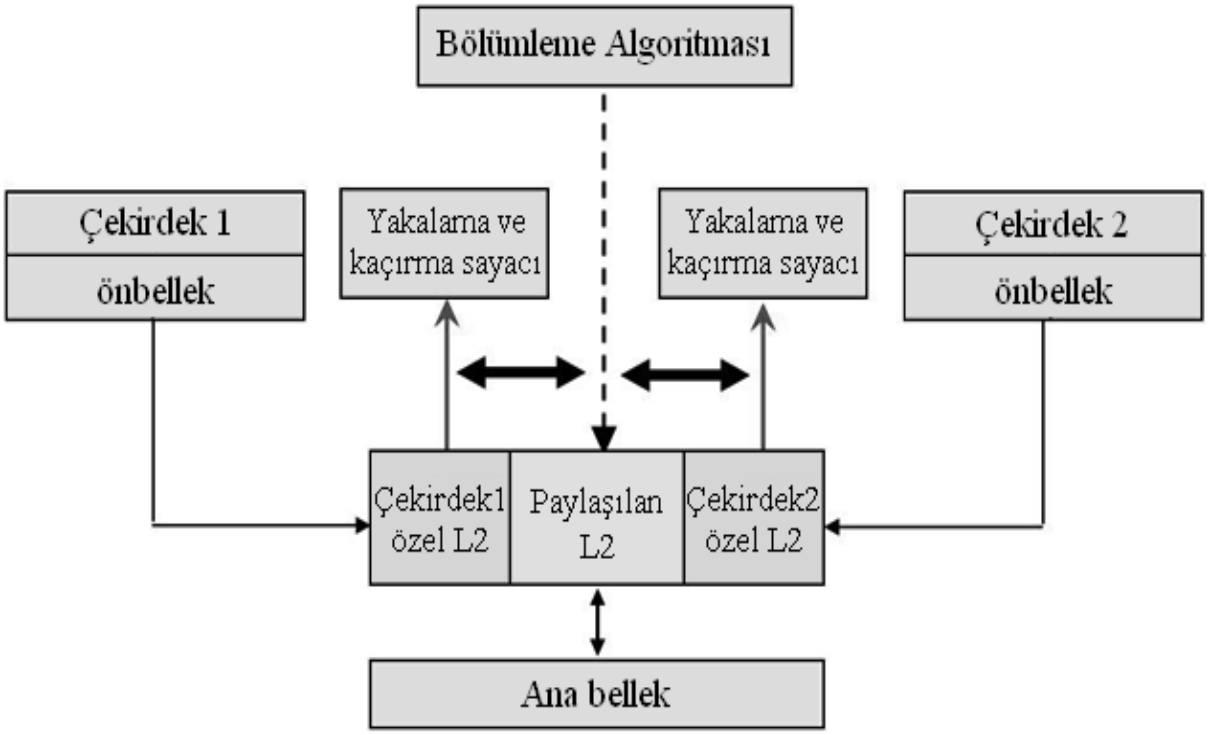
Bu makalede, çok çekirdekli işlemcilerde son seviye önbelleği çekirdekler arasında mantıksal olarak paylaştıran adaptif bir mekanizma önerilmektedir. Bu mekanizmayı, çekirdeklerde koşan uygulamaların bellek gereksinimlerine göre çekirdeklere atanmış önbellek set sayısının artırılması veya azaltılmasını sağlaması nedeni ile Set-Bazlı Bölümleme (SBB) olarak isimlendirdik. SBB, bu bölümleme işini çalışma anında donanım tarafından elde edilmiş periyodik olarak örneklenmiş bellek istatistiklerini inceleyerek yapmaktadır. Özetle, çekirdekler son seviye önbellekte kendilerine ait bölüme sahip olmakta ve önbellek çalınmasından kaynaklanan sorunlar azalmaktadır.

Literatürde yol-bazlı önbellek bölümlenmesi öneren birçok çalışma bulunmaktadır. Ancak, set-bazlı önbellek bölümlenmesi öneren çalışmaların sayısı çok azdır. Bunun en önemli nedeni, yol-bazlı bölümlemenin set-bazlı bölümlemeye göre nispeten kolay ve anlaşılır olmasıdır. Bunun yanında, set-bazlı bölümlemenin yol-bazlı bölümlemeye göre birtakım avantajları mevcuttur. Bu çalışmadaki motivasyonu daha iyi açıklayabilmek için bu avantajlar aşağıda listelenmektedir:

—Son seviye belleğin daha iyi çözünürlükte paylaşılması: Önbelleklerde yol sayısı set sayısına kıyasla çok daha azdır. Önbellekler, çekirdekler arasında yol-bazlı paylaştırıldığında paylaşılan minimum boyut, set sayısı kadar önbellek bloku olabilir. Bu durumda önbellek ihtiyacı çok az olan bir çekirdeğe çok fazla sayıda blok atamak veya hiç önbellek vermemek arasında tercih yapılması gerekecektir. Ayrıca, iki çekirdek arasında yol paylaşımı konusunda stabilizasyon sorunları ve osilasyonlar set-bazlı paylaşıma göre çok daha fazla görülecektir.
—Ölçeklenme: Önbelleklerdeki yol sayılarının azlığı gelecekteki çok fazla çekirdekli mimarilere geçişte yol-bazlı paylaşılan önbelleklerde sorunlara yol açacaktır. Tipik bir son seviye önbellekteki yol sayısı 8 veya 16'dır. Bu durumda 16 çekirdekli bir işlemcide her bir çekirdeğe 16 yollu bir önbellekte 1 yol düşmektedir. Böyle bir mimarinin 8 yollu bir önbellek ile desteklenmesi ise mümkün değildir. Set-bazlı bölümleme algoritmalarının ise bu tip bir sorunu yoktur.
—Önbellek yer değiştirme algoritmasında özgürlük ve önbellek organizasyonu yapısını koruma: Önbellek yolları çeşitli uygulamalara ve dolayısıyla çekirdeklere atandıklarında önbelleğe ait mevcut yer değiştirme algoritmalarının ve organizasyonunun kullanılması mümkün olmamaktadır. Set-bazlı algoritmaların ise uygulandıkları önbellek üzerinde bu tür yan etkileri yoktur. Önerdiğimiz mekanizma, doğrudan-eşlemsel (direct-mapped) veya tamamen-çağrışımsal (fully associative) önbelleklerde rahatlıkla kullanılabilir.
—Minimum boyutta önbellek eklentisi: Yol-bazlı bölümleme yöntemlerinde her önbellek yolu için çok sayıda sayaç ve bu bilgilerin kontrol mekanizmasına iletilmesi için bir o kadar da yola ihtiyaç duyulmaktadır. Set-bazlı bir algoritmada ise bu sayaçların ve yolların sayısı çekirdek sayısı ile sınırlanmıştır.

Önbellekler, bellek duvarı sorunu ile mücadelede çok etkili yapılardır. Eş-zamanlı Çoklu İş parçacıklı (EÇİ) ve Çok Çekirdekli işlemciler (ÇÇİ), iş parçacığı seviyesinde paralelizmi açığa çıkararak yüksek iş çıktısı sağlamayı hedefler. Bu mimarilerin karşılaştığı en önemli sorunlardan birisi, son seviye önbellekler gibi paylaşılan özkaynakların yüksek iş çıktısı sağlarken adil şekilde paylaşılmasıdır.

Herhangi bir iş parçacığının son seviye önbellek erişiminin çalışan diğer iş parçacıklarının erişimleri nedeni ile tamamen etkisizleşmesi tehlikesini ortadan kaldırmak için en basit çözüm statik bölümleme olarak bilinmektedir. Bu yaklaşımda, paylaşılan önbellek, çekirdekler arasında eşit olarak paylaştırılmaktadır. Ancak, statik bölümleme sonucunda çok fazla bellek erişim ihtiyacı duyan uygulamalar istedikleri boyutta önbelleğe sahip



Şekil 1. SBB Önbelleği (SBP Cache)

olamadıkları gibi hiç bellek erişimi ihtiyacı duymayan uygulamalara da ihtiyaçlarının ötesinde önbellek sunulmaktadır.

Paylaşılan önbellek yapılarının dinamik olarak bölümlenmesi ilk olarak Suh ve ekibi tarafından ortaya atılmıştır [1][2]. Bu çalışmalarda önerilen mekanizma, birtakım donanım sayaçları kullanan, sisteme çok fazla yük getirmeyen bir bellek izleme sistemine dayalıdır. Bu sistemdeki sayaçlar, önbellek boyutu arttıkça önbellek yakalamalarındaki marjinal kazancı göstermektedir.

Stone ve ekibi, önbelleğin dinamik bölümlenmesi üzerinde çalışırken uygulamaların ihtiyaçlarına göre önbellek bölümlemesi yapan, En Eski Ulaşılan (EEU) yer değiştirme politikasının en iyi politika olmadığına inanıyordu [3]. Bu çalışmanın ardından, Chiou ve ekibi, her iş parçacığının profil bilgisine dayalı olarak çalışan iş parçacıklarına önbellek yollarını atamayı önermiştir [4]. Benzer şekilde, Settle ve ekibi, dinamik, yol-bazlı önbellek bölümlenmesini araştırmaktadırlar [5].

Lin ve ekibi, sayfa renklendirme adındaki işletim sistemi tekniğine dayalı bir önbellek bölümleme algoritması önermektedir [6]. Bir sayfa rengi, önbellek endeksi ile fiziksel adresin fiziksel sayfa numarasının çeşitli ortak bitlerinden oluşturulur. Ardından, fiziksel adreslenebilir önbellek, sayfa renkleri kullanılarak çakışmayan bölgelere ayrılarak, aynı renkli sayfaların da o renge sahip bölgeye erişmesi sağlanır.

Qureshi ve Patt, yol-bazlı önbellek bölümleme üzerine yaptıkları çalışmada, sisteme çok az ek yük oluşturan ve verilen önbellek miktarına göre ilgili uygulamaların önbellek kaçırmalarını izleyen donanımsal bir devre önermektedir [7]. Moreto ve ekibi de benzer, yol-bazlı başka bir çalışmada işlemci performansına aşırı derecede etkisi olan izole L2 kaçırmalarına daha çok, kümelenmiş L2 kaçırmalarına ise daha az puan veren bir algoritma önermektedir [8][9]. Puanlama işlemi, yardımcı etiket dizini, kaçırma ve yakalama durumu tutan yazmaçlar içeren ek bir donanım ile yapılmaktadır.

Son olarak, Sanchez ve Kozyrakis, *Vantage* adını verdikleri önbellek bölümleri arasında yüksek çağrışım ve güçlü izolasyon sağlayan bir metod önermektedir [10]. Yazarlar, her bir önbellek bölümünün boyutunu o bölüme giren ve çıkan satırların ortalama hızını dengelemeye çalışarak sağlamaktadır. Vantage, en iyi Zcache adı verilen bir önbellek mimarisi ile çalışmaktadır [11].

Makalenin geri kalan kısmı şu şekilde planlanmıştır. İkinci kısımda önerilen mekanizmanın detaylarını açıklamakta, ardından gelen üçüncü kısımda ise izlenen deneysel yöntem açıklanmaktadır. Dördüncü kısımda benzetim sonuçları sunulmakta ve değerlendirilmektedir. Son olarak, beşinci kısımda çalışmamızı sonuçlandırmaktayız.

2. SET-BAZLI ÖNBELLEK BÖLÜMLENMESİ (SET-BASED CACHE PARTITIONING)

Bu kısımda, önerilen mekanizmanın tasarım ve gerçekleme detayları açıklanmaktadır. Bu çalışma, özellikle çift çekirdekli ÇÇİ mimarisine odaklanmıştır. Öncelikle, önerilen mekanizma, son seviye önbelleğe önbellek erişimleri ve önbellek kaçırmalarını tutan donanım sayaçları eklemektedir. Ayrıca, bu sayaçların yanında, set-bazlı önbellek bölümlemesi yapan bir kontrol devresi de ek donanım olarak önerilmektedir. Şekil 1'de bölümleme algoritmasının son seviye önbellek üzerindeki parçaların boyutlarının dinamik olarak ayarlanmasına karar verişi gösterilmektedir.

2.1. Bölümleme Algoritması (Partitioning Algorithm)

Şekil 2'de sunulan Bölümleme algoritmasının ana hedefi, her bir çekirdek için önbellek kaçırma oranlarını azaltarak baz işlemciye oranla daha iyi işlemci performansı elde etmektir. Algoritmadaki döngü içinde yeni bir bölümleme adımına ulaşana kadar her bir çekirdeğe ait önbellek erişim ve kaçırma sayıları toplanmaktadır. Her bölümleme adımında, çekirdeklere ait önbellek kaçırma oranları şu şekilde hesaplanmaktadır:

$$\text{Dinamik Kaçırma Oranı}_{\text{aralık}} = \frac{\text{Kaçırma sayısı}_{\text{aralık}}}{\text{Erişim Sayısı}_{\text{aralık}}} \quad (1)$$

Burada, Dinamik Kaçırma Oranı (DKO) ilgili zaman aralığı için önbellek kaçırma oranıdır. Sonuç olarak, DKO, ilgili çekirdeğe ait kümülatif kaçırma oranından daha ilgi çekici olan anlık kaçırma oranını sunmaktadır. DKO, çalışan uygulamaların anlık bellek gereksinimlerinin takip edilmesi için yararlı bir parametredir.

Algoritmada son adım, çekirdekler arasındaki Dinamik Kaçırma Oranı Farkının (DKOF) hesaplanmasıdır. Bu parametre, ikinci denklemdeki şekilde hesaplanmaktadır:

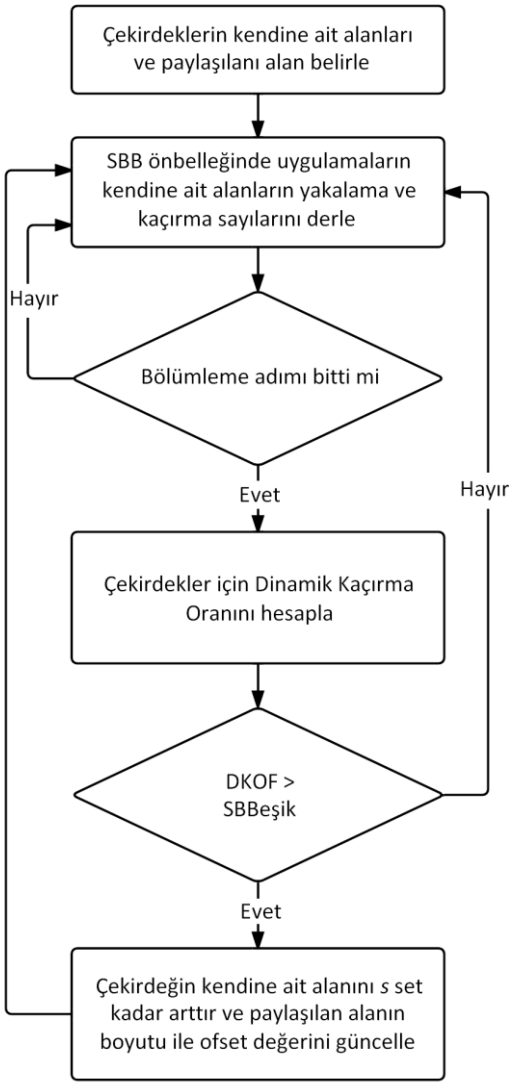
$$\text{DKOF} = \left|\text{DKO}_{\text{çekirdek0}} - \text{DKO}_{\text{çekirdek1}}\right| \quad (2)$$

DKOF değeri, belirli bir eşik değerinin (Set-Bazlı Bölümleme eşik değeri, SBBeşik) altında olduğu durumda mevcut konfigürasyon korunur; aksi takdirde, daha yüksek DKO değerine sahip çekirdeğe *s* değeri kadar yeni set eklenir. Bunlara ek olarak, ilgili çekirdeğe ait ofset değeri ve ortak bölüme ait boyut güncelleme işlemi Set-Bazlı Bölümleme Adres Tercüme Devresi (SBBATD) ile gerçekleştirilir.

2.1.1. SBB Adres Tercüme Devresi (SBP Cache Address Translation Logic)

Kısaca SBBATD olarak adlandırılan bu devre, önbelleğe ulaşan fiziksel bir adresin ilgili çekirdeğe ait önbellek bölümüne yönlendirilmesini ve erişimini sağlar. Şekil 3'te SBB önbelleğin ve adres tercüme devresinin mantıksal yerleşimi gösterilmektedir. Örneğin, birinci çekirdeğe ait bölüme erişilmesi için, herhangi bir fiziksel adres, SBBATD tarafından birinci çekirdeğin sınırlarındaki 0 ve 499 setleri arasına yönlendirilmektedir.

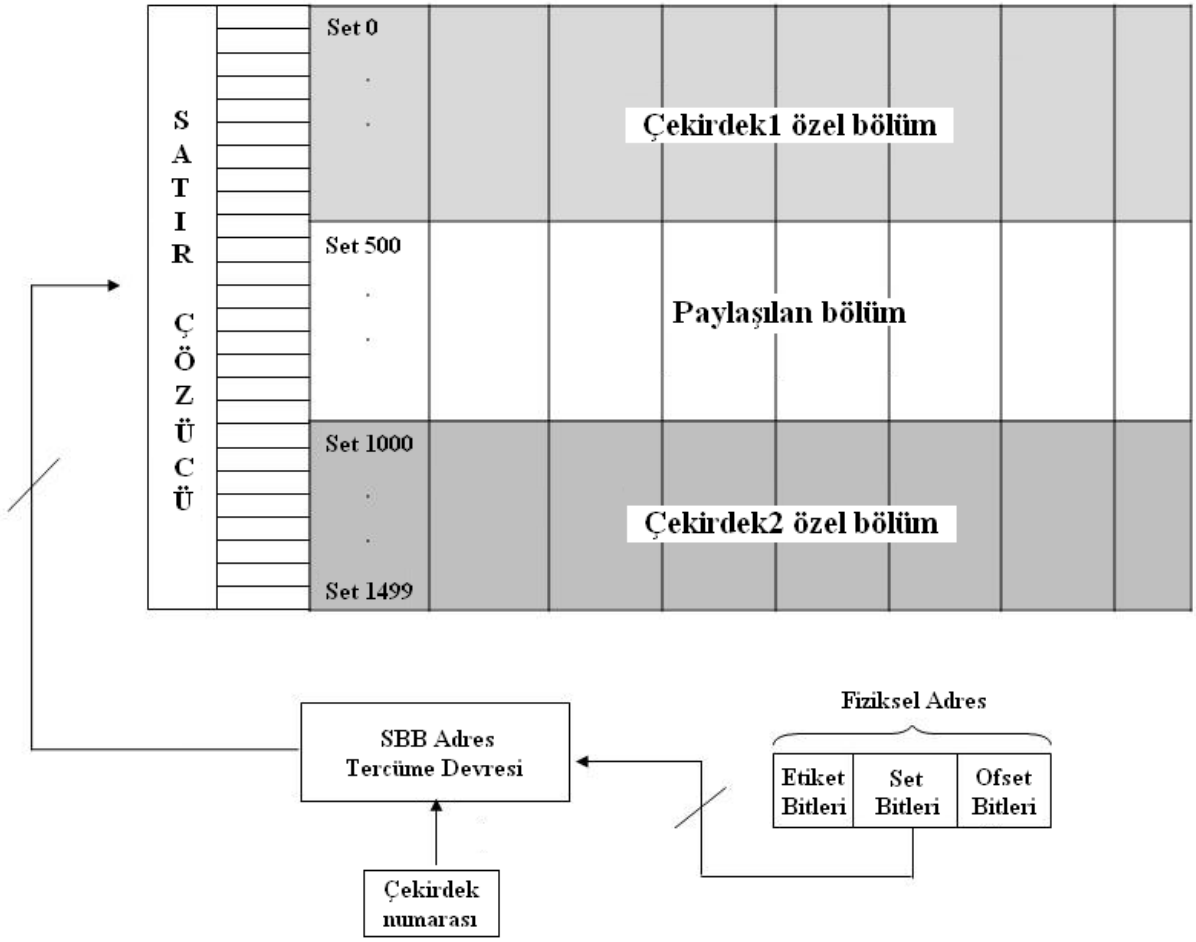
$$\text{set}_{\text{çekirdek}} = (\text{set}_{\text{orijinal}} \bmod \text{Bölüm Boyutu}_{\text{çekidek}}) + \text{ofset}_{\text{çekirdek}} \quad (3)$$



Şekil 2. Set bazlı bölümleme algoritması (Set based partitioning algorithm)

2.2. SBB Eşik Değeri (SBP Threshold Value)

Bu çalışmada önerilen bölümleme algoritması, çekirdekler arasındaki DKOF değerine bağımlıdır. DKOF değerinin uygulamaların davranışını takip edebilecek uygun bir eşik değeri ile karşılaştırılması ise çok önemlidir. Bu eşik değeri, statik veya dinamik karakteristikte olabilir. Çalışmamızın başında, eşik değerini statik olarak %5 ve %10 değerleri ile test etmiş bulunmaktayız. Ancak sabitlenen eşik değeri, bütün uygulama karışımlarında hedeflenen performans değerlerine ulaşılmasına izin vermediği



Şekil 3. SBB önbelleğinde SBBADT örneği (SBPATL example in SBP cache)

için, eşik değerinin dinamik olarak değişmesine izin vererek çalışan uygulamaların davranışına göre kendini ayarlayabilen Dinamik Eşik Mekanizması (DEM) önermekteyiz.

2.2.1. Dinamik Eşik Mekanizması (DEM)
(Dynamic Threshold Mechanism (DTM))

DEM, DKOF değerlerini belirli bir zaman penceresinde tutarak, ortalama DKOF değerini SBBeşik değeri olarak seçmektedir. Bu şekilde, uygulamaların ihtiyaç duydukları önbellek miktarına daha çabuk ve doğru şekilde sahip olabildikleri elde edilen sonuçlardan da görülmektedir. Ortalama DKOF değerinin elde edilmesi için, her bölümleme adımındaki DKOF değerinin kayıt edilmesine ve *n* tane DKOF değerini tutabilen bir yapıya ihtiyaç duyulmaktadır.

2.3. Hibrit Son Seviye SBB Önbellek
(Hybrid Last Level Unified Cache)

SBB önbellekte, her çekirdek öncelikle kendine ait bölüm üzerinde arama yapmaktadır. Aranan blokun burada bulunması durumunda standart önbellek yakalama süresi kadar zaman harcanır. Kaçırma durumunda ise, aynı önbellek üzerinde bu sefer paylaşılan bölümünde arama yapılır. Aranan blokun burada bulunması durumunda standart önbellek yakalama süresinin iki katı süre harcanmış olacaktır.

3. DENEYSEL YÖNTEM
(EXPERIMENTAL METHODOLOGY)

Tasarımımızın başarısını ölçebilmek için, çok çekirdekli Alpha işlemci modelini de barındıran, tüm işlemci özkaynaklarını çevrim doğruluğunda (cycle-accurate) modelleyen ve işlemci başarımından, güç tüketim değerlerine kadar birçok istatistiki bilgiyi de sunan çoklu iş parçacıklı simülasyon ortamı olan M-Sim [12] kullanılmıştır. Çalışmamızda, iki çekirdekli bir işlemci simülasyonu yapılmıştır. Çalışmamız boyunca yalnızca önbellek ile ilgili olan parametrelerde deneysel değişiklikler yapılmıştır; işlemcilerin geri kalan tüm kısımlarında ise Tablo 1'deki değerler kullanılmıştır.

Her çekirdek, kendine ait birinci seviye veri ve buyruk önbelleği kullanmaktadır. L2 önbellek ise son seviye olarak düşünülmüştür ve çekirdekler arasında paylaşılmaktadır. Detaylı bellek parametreleri Tablo 2'de gösterilmiştir.

Karmaşıklık ek yükünü tahmin edebilmek için, tasarımımızda kullanılan tüm donanım parçaları için gerekli transistör sayısının yaklaşık hesabı yapılmıştır. Daha sonra, 2MB paylaşılan bir önbellek oluşturmak için gerekli transistör sayısı hesaplanarak, tasarımımızın gerçeklenmesinin %0,01 gibi göz ardı edilebilecek minimal bir karmaşıklık artışına sebep olacağı hesaplanmıştır.

Çalışmamızı değerlendirmek için SPEC2K uygulamaları kullanılmıştır. Testler için, son seviye önbellek iki çekirdek arasında paylaşıldığından, iki uygulamadan oluşan karışımlar hazırlanmıştır. Çalışmamızda çeşitli karışımlar kullanabilmek için, iş yükleri karakteristiklerine göre 4 sınıfa ayrılmıştır: 1) Çok az bellek erişim ihtiyacı duyan (C), 2) Çok fazla bellek erişim ihtiyacı duyan (M), 3) Bir adet çok az ve bir adet çok fazla bellek erişim ihtiyacı duyan (H1) ve son olarak 4) Bir adet çok fazla ve bir adet orta derecede bellek erişim ihtiyacı duyan uygulamadan oluşan melez iş yükleri (H2).

Tablo 1. Çekirdek parametreleri (Processor core specifications)

Çalıştırılan buyruk sayısı	200 Milyon
Atlanılan buyruk sayısı	50 Milyon
Çekirdek başına düşen uygulama sayısı	1
Süperskalar genişliği	8 buyruk/çevrim
Çalıştırma kuyruğu boyutu	64
Yükleme/kaydetme (load/store) kuyruk boyutu	48
Yeniden sıralama kuyruk boyutu	256
Fiziksel yazmaç dosyası boyutu	256
Tamsayı matematiksel mantık birimi sayısı	8
Tamsayı çarpım/bölüm devresi sayısı	3
Kayan noktalı matematiksel mantık birimi sayısı	8
Kayan noktalı çarpım/bölüm devresi sayısı	3

Bu sınıflandırmanın sonucunda 20 karışım belirlenmiştir. Detaylar, Tablo 3'te gösterilmiştir. Uygulamalar 200 milyon buyruk boyunca çalıştırılmıştır. Çalışmamızda kullanılan uygulamalar ve detayları ise Tablo 4'te sunulmaktadır.

Tablo 2. Bellek parametreleri (Memory Specifications)

Çekirdeğe özel L1 buyruk ve veri önbelleği	32KB, 32B blok, 512 satır, 2-yol,EEU
(Unified) Paylaşılan L2 önbellek	2MB, 128B blok, 2K satır, 8-yol,EEU
L1 buyruk ve veri önbelleği yakalama süresi	1 çevrim
L2 önbellek yakalama süresi	20 çevrim
Bellek erişim gecikme süresi	300 çevrim
Bellek erişim yol genişliği	8 byte
Bellek kapısı sayısı	2
Devredilen önbellek boyutu	2, 4, 8, 128
Bölümleme adımı uzunluğu	(100K, 1M, 3M 5M, 15M) çevrim
DEM kuyruk büyüklüğü	5, 10, 25

Tablo 3. İş yükleri (Workload types)

Sınıf	İş Yükü
C-tipi	(ammp-swim),(fma3d-perl),(wupwise-sixtrack),(wupwise-perl),(swim-fma3d)
M-tipi	(vpr-apsi),(twolf-bzip2),(parser-twolf),(mgrid-apsi),(mesa-mgrid)
H1-tipi	(fma3-mgrid),(perl-parser),(parser-fmad3),(gzip-parser),(swim-twolf),(twolf-ammp)
H2-tipi	(mgrid-art),(art-mesa),(apsi-art),(parser-art)

4. SİMÜLASYON SONUÇLARI (SIMULATION RESULTS)

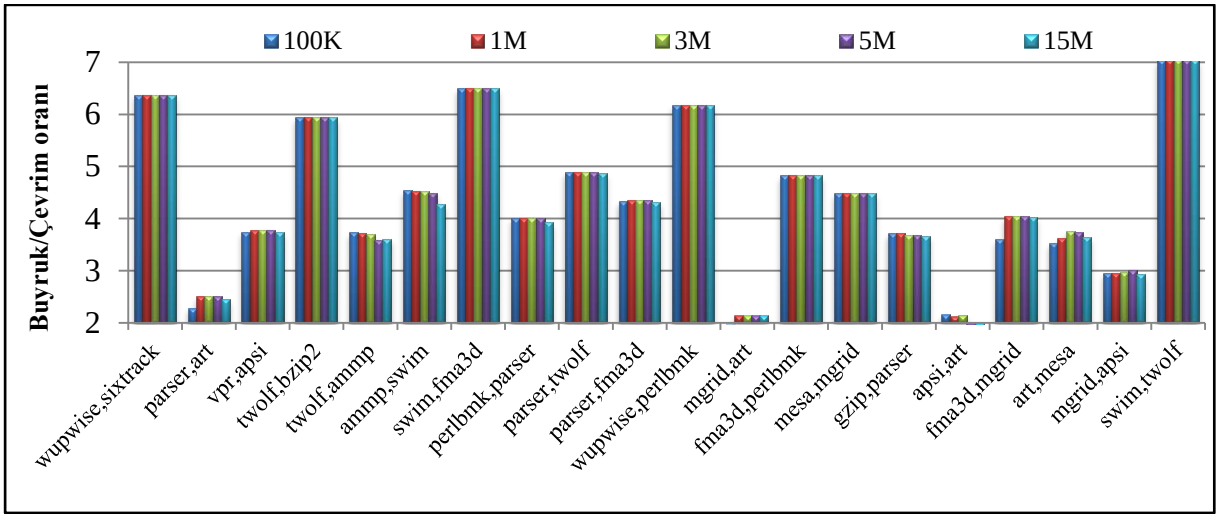
Tasarımımızın performans sonuçları iki farklı önbellek ile karşılaştırılmıştır. Bunlardan ilki, çalışan uygulamaların önbellek girdileri için yarıştığı baz işlemci önbelleği iken, ikincisi her uygulamaya sabit miktarda ön bellek girdisinin ayrıldığı statik bölümlemedir.

4.1. Bölümleme Adımı Uzunluğunun SBB Önbelleği Üzerindeki Etkisi (SBP Cache With Different Update Intervals)

Bölümleme adımı, bölümleme algoritmamızın çalışmaları arasında geçen süreyi belirtmektedir. Bu parametre için çeşitli değerler test edilerek performans üzerindeki etkileri Şekil 4'te detaylı bir şekilde sunulmaktadır. Şekil 4'teki performans sonuçları;

Tablo 4. Uygulama özellikleri (Application specifications)

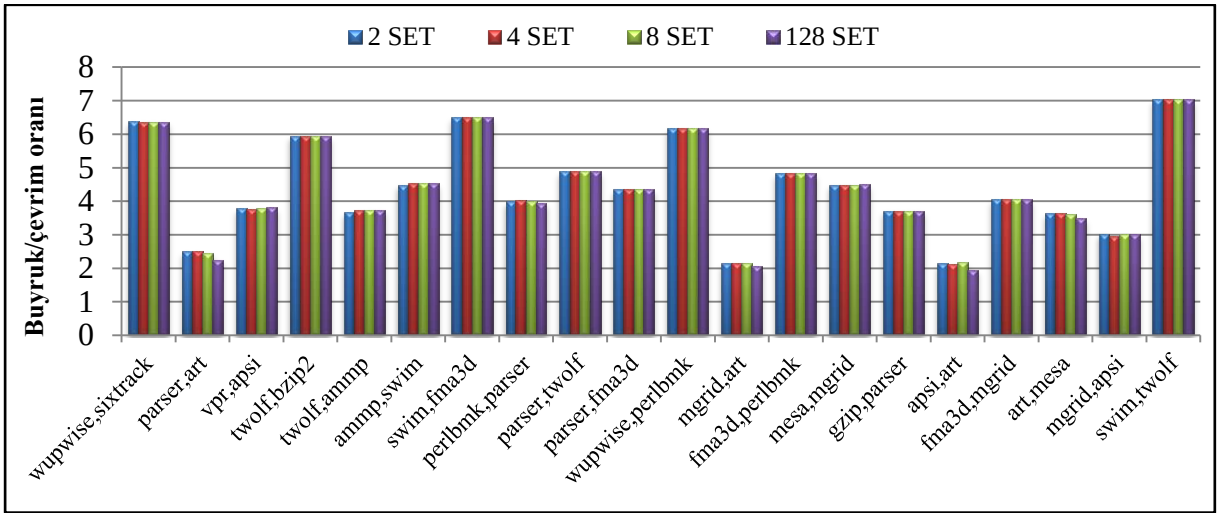
Uygulama	Bellek Buyruğu %	Tür	Açıklama
gzip	%50	Tamsayı	Sıkıştırma
vpr	%30	Tamsayı	FPGA Devre Yerl. ve Yönlendirme
parser	%55	Tamsayı	Kelime İşleme
perlbmk	%39	Tamsayı	PERL Programlama Dili
twolf	%35	Tamsayı	Yerl. ve Yönlendirme Simülatörü
wupwise	%20	Kayan noktalı	Fizik / Kuantum Kromo Dinamiği
swim	%30	Kayan noktalı	Sığ Su Modellemesi
mgrid	%45	Kayan noktalı	Çoklu Izgara Çözücü: 3B Pot. Alan
mesa	%50	Kayan noktalı	3B Grafik Kütüphanesi
art	%42	Kayan noktalı	Resim Tanıma / Sinir Ağları
ammp	%38	Kayan noktalı	Hesaplamalı Kimya
fma3d	%30	Kayan noktalı	Sonlu Eleman Çarpışma Simül.
sixtrack	%35	Kayan noktalı	Yüksek Enerjili Nükleer Fizik
apsi	%49	Kayan noktalı	Meteoroloji/ Kirli Madde Dağılımı



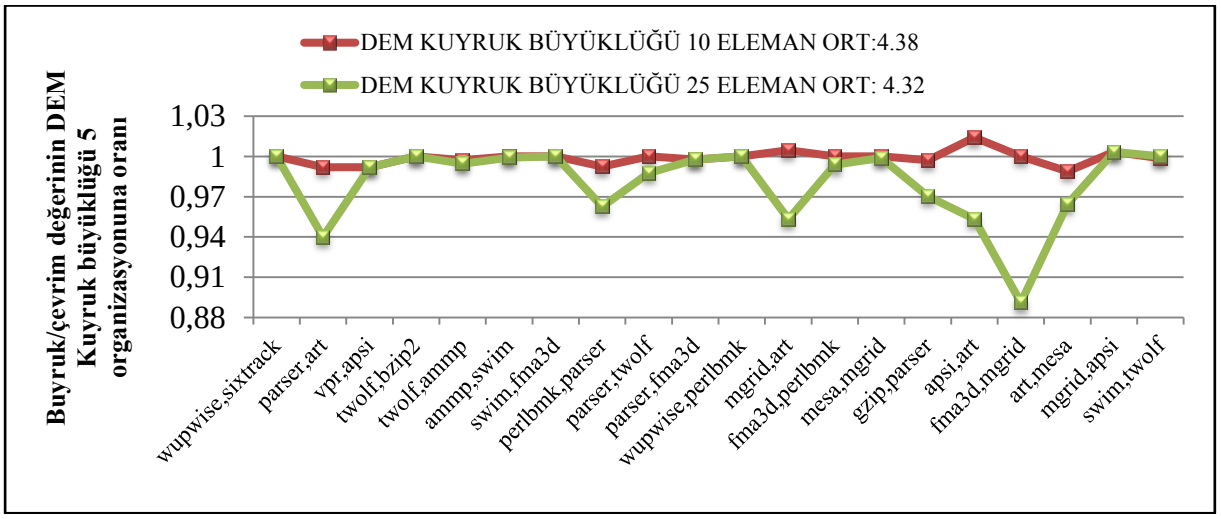
Şekil 4. SBB önbelleğinin çeşitli bölümleme adımı uzunluğu değerlerinde gösterdiği performans (SBP cache performance with different update intervals)

SBB önbelleğinin ortalamada, bölümleme adımının uzunluğu 1M ve 3M çevrim değerini taşırken 100K gibi küçük bir değerde verdiği sonuçlardan %1.1 daha iyi sonuç verdiğini göstermektedir (özellikle (parser, art), (fma3d, mgrid) ve (mgrid, art) iş yüklerinde). Bu sonuçlar, bölümleme algoritması çok sık çalıştırıldığı zaman performans düşüşünün kaçınılmaz olduğunu göstermektedir. Bu durum şu şekilde açıklanabilir: uygulamanın davranışı istikrarlı olduğunda, önbellek boşaltılmasından mümkün olduğunca kaçınmak gerekmektedir. Ancak önbellek yeniden bölümlendiğinde, bölümler zorunlu olarak boşaltılmaktadır ki bu da bu bölümlerde önbellek yakalamalarının kaybedilmesi anlamına gelmektedir.

5M veya 15M gibi büyük bölümleme adımı uzunlukları göz önüne alındığında ise ortalama %1.6 performans düşüşü kaydedilmiştir. Bazı iş yüklerinde ise performans düşüşü yüksek oranlara varmaktadır. Örneğin (ammp, swim) ve (apsi, art) iş yüklerinde bu kayıplar %5.7 ve %17 oranlarına varmaktadır. Bölümleme adımı uzunluğu çok fazla olduğunda, bölümleme algoritması, uygulamadaki ani davranış değişikliklerine uyum sağlayamayabilir. Bir uygulamanın davranış biçimi değiştiğinde ve bölüm boyutları uygulamanın anlık ihtiyaçlarına uygun olmadığında önbellek kaçırması oranı artacak ve dolayısıyla performans düşecektir. Bölümleme adımı uzunluğunun daha da fazla olduğu uç durumlarda ise, SBB performansı statik bölümleme performansına yakın olacaktır.



Şekil 5. SBB önbelleğinin çeşitli devredilen önbellek boyutu değerlerinde gösterdiği performans (SBP cache performance with various resizing amounts)

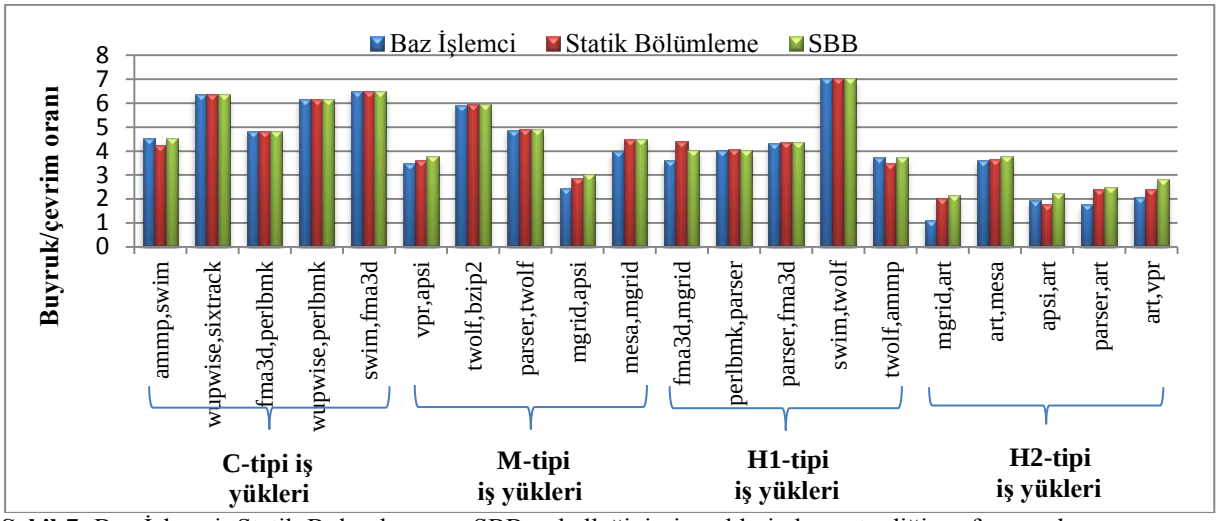


Şekil 6. SBB önbelleğinin çeşitli DEM kuyruk büyüklüğü değerlerinde gösterdiği performans (SBP cache performance with different queue sizes)

4.2. SBB Önbelleği ve Devredilen Önbellek Boyutu (SBP Cache and the Resizing Amount)

Bu noktada, çalışmamızdaki asıl amacımızın, önbelleği mantıksal ve dinamik olarak, set-bazlı bir biçimde bölümlemek olduğunu vurgulamak isteriz. Bu bölümde, her bölümlemede uygulamalara verilen ya da onlardan alınan set sayısını belirleyen devredilen önbellek boyutu parametresinin etkileri incelenmektedir. Şekil 5, SBB önbelleğinin, bölümleme adımı uzunluğu 1M çevrim iken; 2, 4, 8 ve 128 setlik devredilen önbellek boyutu değerleri için gösterdiği performansları göstermektedir.

Şekil 5'teki sonuçlar, set-bazlı önbellek bölümlemenin gücüne işaret etmektedir. SBB önbelleğinde 128 setlik bir devredilen önbellek boyutu, 16 önbellek yolu sahibi bir önbellekte yol-bazlı önbelleğin sahip olabileceği asgari devredilen önbellek boyutuna eşittir. Böylelikle, SBB önbelleği ile yol-bazlı önbelleği karşılaştırabilmekteyiz. Ortalamada, SBB önbelleği, yol-bazlı önbellekten test edilen melez iş yüklerinde %2, tüm iş yüklerinde ise %0.7 daha iyi performans göstermektedir. Bu noktada yol-bazlı önbelleğin (parser, art) iş yükünde %10 ve (mgrid, art) iş yükünde %4.2 gibi ciddi performans düşüşleri gösterdiği gözlenmektedir. Bu performans düşüşü, yol-bazlı önbelleğin uygulamalara ihtiyaçlarından fazla önbellek alanı ayırarak kontrol mekanizmasında salınımlar olmasına sebep olmasından kaynaklanmaktadır. Salınımlar devredilen önbellek boyutu ile ilişkilidir. Bu durumda, az ön bellek alanına ihtiyacı olan uygulamalara büyük önbellek alanı ayırmak; örneğin 'art' ve 'parser' gibi çok fazla bellek erişim ihtiyacı duyan uygulamaların kaçırma oranlarının artmasına sebep olacaktır. Bu artış, ileride bölümleme algoritmasının karar verme sürecini de olumsuz yönde etkileyecektir. Sonuç



Şekil 7. Baz İşlemci, Statik Bölümleme ve SBB önbelleğinin iş yüklerinde gösterdiği performanslar (Performance of Baseline Processor, Fixed Partitioning and SBP workloads)

olarak, 'art' uygulamasına verilen önbellek kaynakları kendisinden alınacak ve 'parser' uygulamasına verilecektir. Kontrol mekanizmasındaki bu salınımlar, birçok bölümleme adımı boyunca devam edecek ve işlemcinin performansını düşürecektir.

Öte yandan, yol-bazlı önbellek, bazı çok fazla bellek erişim ihtiyacı duyan iş yüklerinde SBB önbelleğinden daha yüksek performans göstermektedir. Örneğin, SBB önbelleğinin performansında (vpr, apsi) iş yükünde %1.1'lik ve (mesa, mgrid) iş yükünde %0.4'lük bir düşüş gözlenmiştir. Çok fazla bellek erişim ihtiyacı duyan bir uygulama bir anda çok sayıda önbellek veri bloğuna ihtiyaç duyduğunda, bu ihtiyaçları anında karşılayabilen yol-bazlı önbellek, devredilen önbellek boyutu düşük diğer organizasyonlardan daha iyi sonuçlar verebilmektedir.

4.3. DEM Kuyruk Büyüklüğünün SBB Önbelleği Üzerindeki Etkisi (Effect of DTM Queue Size on SBP Cache)

Bölüm 2'de Dinamik Eşik Mekanizmasını ve SBB önbelleğindeki rolünü açıklanmış, uygulamalar arasındaki kaçırma farkını barındıran bir kuyruk yapısıyla gerçeklenebileceği gösterilmişti. Bu bölümde ise DEM kuyruk büyüklüğünün SBB önbelleğinin performansı üzerindeki etkisi incelenmektedir.

Şekil 6'da DEM kuyruk büyüklüğü 5, 10 ve 25 iken SBB önbelleğinin performansı gösterilmektedir. DEM kuyruk büyüklüğü değeri 25 iken, 5'e kıyasla, melez iş yüklerinde ortalama %3.3'lük ve tüm iş yüklerinde ortalama %1.5'lik bir düşüş görülmektedir.

4.4. SBB Önbelleğinin Çeşitli İş Yüklerindeki Performansı (SBP Cache Results for Different Workloads)

Çok az bellek erişim ihtiyacı duyan iş yüklerinde; SBB önbelleği, baz işlemci önbelleği ve statik bölümlenmiş önbellek birbirlerine oldukça yakın performanslar sergilemektedir. Bu tip iş yüklerinin bellek erişimleri çok yüksek olmadığından ve verilen önbellek alanı bu iş parçacıklarının yüksek performans göstermesi için yeterli olduğundan; bu beklenen bir sonuçtur. Bu organizasyonların performansları Şekil 7'de görülmektedir.

Çok az bellek erişim ihtiyacı duyan iş yüklerinin aksine, çok fazla bellek erişim ihtiyacı duyan iş yükleri, ani ihtiyaçlarını karşılayabilen, verimli bir önbellek organizasyonuna ihtiyaç duyarlar. Bu tip iş yüklerinde, SBB önbelleği ve statik bölümlenmiş önbellek; uygulamalara kendine ait alanlar tahsis ederek veri kirlenmesini önlediğinden birbirlerine yakın performanslar göstermektedir. SBB önbelleği, çok fazla bellek erişim ihtiyacı duyan iş yüklerinde ortalama %9 performans artışı göstermektedir.

SBB önbelleği, (vpr, apsi) ve (mgrid, apsi) gibi bazı iş yüklerinde, bellek kullanımı daha yoğun olan uygulamalara daha çok önbellek alanı sağladığından, statik bölümlenen önbelleğe kıyasla %8.3 ve %24 gibi performans artışları göstermektedir. Çok fazla bellek erişim ihtiyacı duyan iş yükleri için her üç önbellek organizasyonunun performansları Şekil 7'de görülmektedir.

Şekil 7, aynı zamanda H1-tipi iş yükleri için sonuçları göstermektedir. Bu iş yüklerinde, SBB önbelleği paylaşılan önbellek organizasyonuna göre %2.4 daha iyi performans göstermektedir. Öte yandan, (perlbmk, parser) ve (fma3d, mgrid) iş yükleri için statik bölümlenmiş önbellek en iyi sonuçları vermektedir. SBB önbelleğinin bu iş yüklerinde gösterdiği

performans düşüşü, önbellek boşaltılması problemi ile açıklanabilir. Çekirdeklere ek ön bellek setleri atandığında, kendine ait ve paylaşılan bölümlerdeki tüm veri blokları boşaltılmaktadır. Bu boşaltmalar yol bazlı önbelleklerde de olmaktadır, ama etkileri şu ana kadar yayınlanmış makalelerde göz ardı edilmektedir.

H2-tipi iş yüklerinin performans sonuçları da Şekil 7'de görülmektedir. SBB önbelleği, bu iş yüklerinin tümünde diğer organizasyonlardan daha iyi sonuç vermektedir. Grafikteki (apsi, art) iş yükü, statik bölümlenmiş önbelleğin paylaşılan önbellekten daha kötü sonuç verdiğini göstermektedir. Bu, özellikle uygulamalardan biri önbellek kapasitesinin yetersizliğinden kaynaklanan kaçırmalar yaşamaya başladığı zaman mümkündür. SBB önbelleği ise, her çekirdeğe uygun miktarda önbellek alanı sağladığından en iyi performans sonuçlarına erişmektedir. SBB önbelleği, H2-tipi iş yüklerinde ortalama %29.4 performans artışı göstermiştir.

5. SONUÇLAR (CONCLUSIONS)

İkinci seviye önbelleklerin geleneksel organizasyonları, paylaşımsız ya da paylaşımlı, çok çekirdekli işlemcilerin önbellek performansını eniyilemekte başarılı olmamaktadır. Dinamik olarak bölümlendirilen önbellek organizasyonlarının, çok çekirdekli işlemcilerde çekirdeklerin önbellek kaçırma sayısını azaltarak daha iyi performans elde edilmesini sağlaması mümkün olmaktadır.

Bu çalışmada önerilen SBB önbelleği; son seviye önbelleğin setlere dayalı bölümlendirildiği, adaptif bir metottur. Ayrıca, bu organizasyonu gerçeklemek için gerekli olan donanımsal karmaşıklık artışı %0.01 gibi göz ardı edilebilir bir orandadır.

SBB önbelleği, tüm performans testleri göz önüne alındığında; çok fazla bellek erişim ihtiyacı duyan uygulamalardan oluşan iş yüklerinde ortalama %9, melez iş yüklerinde ortalama %15 performans artışı sağlamaktadır. SBB önbelleği, set adreslerinin haritalanmasını değiştirdiğinden, her yeniden boyutlandırma işlemi sonucunda önbellekteki mevcut satırlar erişilemez hale gelmektedir. Ancak bu durumun performans üzerindeki etkisi ihmal edilebilir miktardadır.

SBB önbelleğinde bölümleme adımlarının uzunluğu ne çok büyük, ne de çok küçük tercih edilmelidir. Bu parametrenin çok büyük seçilmesi, SBB önbelleğinin statik bölümleme gibi çalışmasına neden olurken; çok küçük seçilmesi ise sistemi istikrarsızlığa sürükleyebilmektedir.

KAYNAKLAR (REFERENCES)

1. Suh, G. E., Devadas, S., ve Rudolph, L., "A new memory monitoring scheme for memory-aware scheduling and partitioning", **In Proc. of the 8th Int. Symp. on High-performance Computer Architecture**. Washington, DC, 117–128, 2002.
2. Suh, G. E., Rudolph, L., ve Devadas, S., "Dynamic partitioning of shared cache memory", **Journal of Supercomputing**, Cilt 28, No 1, 7–26, 2004.
3. Stone, H. S., Turek, J., ve Wold, J. L., "Optimal partitioning of cache memory", **IEEE Transactions on Computers**, Cilt 41, No 9, 1054–1068, 1992.
4. Chiou, D., Rudolph, L., Devadas, S., ve Ang, B. S., "Dynamic cache partitioning via columnization", **In Proceedings of Design Automation Conference**, 2000.
5. Settle, A., Connors, D., Gibert , E., ve González, A., "A dynamically reconfigurable cache for multithreaded processors", **Journal of Embedded Computing - Issues in Embedded Single-chip Multicore Architectures**, Cilt 2, No 2, 221–233, 2006.
6. Lin, J., Lu, Q., Ding, X., Zhang Z., Zhang, X., ve Sadayappan, P., "Gaining insights into multi-core cache partitioning: Bridging the gap between simulation and real systems", **In Int. Symp. on High-Performance Comp. Architecture**, Salt Lake City, UT, 367–378, 2008.
7. Qureshi, M. K., ve Patt, Y. N., "Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches", **In Proc. of the 39th IEEE/ACM Int. Symp. on Microarchitecture**, Washington, DC, 423–432, 2006.
8. Moreto, M. M., Cazorla, F. J., Ramirez, A., ve Valero, M., "Dynamic cache partitioning based on the MLP of cache misses", **Transactions on High-performance Embedded Architectures and Compilers III**, Berlin, Heidelberg, 3–23, 2011.
9. Moreto M. M., Cazorla, F., Ramirez, A., ve Valero, M., "Explaining dynamic cache partitioning speed ups", **IEEE Comp. Architecture Letters**, Cilt 6, No 1, 1–4, 2007.
10. Sanchez, D., ve Kozyrakis, C., "Vantage: Scalable and efficient fine-grain cache partitioning", **In Proceedings of the 38th Annual International Symposium on Computer Architecture**, New York, NY, 57–68, 2011.
11. Sanchez, D., ve Kozyrakis, C., "The zcache: Decoupling ways and associativity", **In Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture**, Washington, DC, 187–198, 2010.
12. Sharkey, J. J., Ponomarev, D., ve Ghose, K., "M-sim: A flexible, multithreaded architectural simulation environment", Tech. Rep. CS-TR-05-DP01, Department of Computer Science, State University of New York, Binghamton, NY, 2005.